Downloads for TPF Family Products

Sample WS-Security Wrapper Application on z/TPF Enterprise Edition V1.1

Note: Before using this information and the product it supports, read the general information under "NOTICES" in this document.

CONTENTS

This file includes the following information:

1.0 ABOUT THIS README

This readme file will guide you through the process of downloading, installing, and using a sample WS-Security SOAP application on your z/TPF system. This sample application demonstrates how to use the z/TPF SOAP consumer application programming interfaces (APAR PJ35511) to invoke a Web service using WS-Security (XML Encryption) (APAR PJ37673) on a remote platform or on the same z/TPF system.

The sample WS-Security Wrapper package provides you with a complete sample that can be run on your z/TPF system. You can use it as a starting point for your own WS-Security SOAP applications and Web service stubs, use it for training purposes, or use it as-is. Go to the IBM TPF Product Information Center for more details about WS-Security and SOAP consumer support.

Note: The TPF development lab does not maintain this application and will not accept APARs on this code.

2.0 SYSTEM REQUIREMENTS

Before proceeding with these instructions, additional downloads may be required from the TPF Support Web site. Do the following:

1. Ensure that PJ35511 (SOAP Consumer) has been applied to your z/TPF system.
2. Ensure that PJ37673 (WS-Security) and PJ37735 (WS-Security errnos) have been applied to your z/TPF system.

3. Download and install the *Sample SOAP Consumer Application* from http://www-01.ibm.com/software/htp/tpf/maint/toolsztpf.html.
4. Download and install the WS-Security Wrapper Sample application. from http://www-01.ibm.com/software/htp/tpf/maint/toolsztpf.html.

3.0 DOWNLOADING

To download this module, do the following:

1. Click the **Download now** button to download the compressed sample WS-Security Wrapper package (the tarball) to your PC. The name of this package is **WS-Security_wrapper_sample_zTPF.tar.Z**.
2. FTP the tarball to your home directory on your Linux system using binary mode:
   - Open an MS-DOS window and activate FTP by using the following command:
     ftp *your.linux.build.machine.com*
   - Sign in using your user name and password.
   - Set the mode to binary by entering the following command:
     binary
   - Send the file to your Linux system by using the following command:
     send c:*your_path*\WS-Security_wrapper_sample_zTPF.tar.Z WS-Security_wrapper_sample_zTPF.tar.Z
   - Exit FTP by entering the following command:
     bye

3. On your Linux system, create a working directory in your root directory by entering the following command:

   mkdir ~/your_workdir

4. Change to the working directory and extract the program files from the WS-Security sample package by entering the following command:

   cd ~/your_workdir
   tar -xzkf ../WS-Security_wrapper_sample_zTPF.tar.Z

   After you have completed this step, you will have the following files on your Linux system in the directory *~/your_workdir:*

   - Consumer files:
     - Sample WS-Security consumer Web service deployment descriptor for the sample application (base/tpf-ws/SecureCalculatorService_consumer.xml)
     - Sample WS-Security consumer extension file descriptor for the sample application (base/tpf-ws/ext/WS-Security/calculatorConsumer_ext.xml)
   - SOAP Message Handler files:
     - Sample WS-Addressing SOAP message handler deployment descriptor for the sample application (base/tpf-ws/WS-Addressing.xml)
   - Provider files:
     - Sample WS-Security user exit code:
       - local_mod/base/rt/cwscue.c
       - local_mod/base/rt/cso9.c
     - Sample z/TPF Web wrapper application code:
       - soap/calculator/soap/csob.c
       - soap/calculator/soap/csob.mak
     - Sample WS-Addressing SOAP message handler code:
       - soap/ws-addressing/csok.c
       - soap/ws-addressing/csok.mak
     - Sample WS-Security provider Web service deployment descriptor for the sample

application (base/tpf-ws/SecureCalculatorService.xml)
        - Sample WS-Security provider extension file descriptor for the sample application (base/tpf-ws/ext/WS-Security/calculatorProvider_ext.xml)
    - Sample XML documents:
        - Encrypted XML request (sampleDocs/SecureCalculatorConsumerRequest.xml)
        - XML request after decryption (sampleDocs/requestAfterDecryption.xml)
        - CalculatorService response before encryption (sampleDocs/responseBeforeEncryption.xml)
        - Encrypted XML response (sampleDocs/SecureCalculatorConsumerResponse.xml)
    - This readme (WS-Security_readme.htm).

4.0 WS-SECURITY

There are many ways to set up and customize WS-Security on your system.

The following are instructions on how this sample WS-Security application is customized.

4.1 Client identification

The Web services client system identification user exit function `tpf_soapGetClientID`, in segment cwscue.c, is called as part of the WS-Security SOAP Message Handler processing. Encrypted inbound SOAP messages into z/TPF have to include information about the sending system identity so that the WS-Security SOAP message handler can select the correct encryption key to do the decryption.

This sample makes use of the WS-Addressing specification to provide this capability. It is up to the end-user to select and implement the client system identification mechanism to be used in your installation. To learn more about how this sample has partially implemented WS-Addressing support, do the following:

1. Open the sample encrypted XML request file called SecureCalculatorConsumerRequest.xml.
2. The sample cwscue.c code expects the inbound SOAP request to contain a `wsa:Address` element in the SOAP Header block. The value of the "Address" is the client identifier and can be any agreed upon name between the client and provider.

   The sample code uses the z/TPF XML API tpf_getElementsByTagName() to extract the "Address" element for the client identifier.

   Locate the "Address" element in the request file that was opened in Step #1:

   ```
   <wsa:From>
        <wsa:Address>http://www.ibm.com/localTPF/clientApplication</wsa:Address>
   </wsa:From>
   ```

   The value of the element called Address (shown above) "http://www.ibm.com/localTPF/clientApplication" will be stored and used for the client identification of this request.

3. If you choose another method for client identifier, open the cwscue.c file to edit the `tpf_soapGetClientID` function to return your system client identification.
4. Helper functions `TPF_SOAP_MSGCTX_GET_CLIENT_ID(ctx)` and `TPF_SOAP_MSGCTX_SET_CLIENT_ID(ctx, client_id)` are provided to manually retrieve and set the client system identifier. The `client_id` must be a NULL terminated string.

4.2 WS-Addressing SOAP Message Handler:

The sample WS-Addressing SOAP message handler for using WS-Addressing along with XML encryption is segment csok.c. It provides the mechanism to identify the consumer that is sending in

Web service requests and the provider that is sending the Web service responses.

- *Outbound requests*

The WS-Addressing handler will add the wsa-addressing elements to the outbound consumer XML request. The elements are added after the WS-Security Header element. (SecureCalculatorConsumerRequest.xml)

```
<wsa:From>
     <wsa:Address>http://www.ibm.com/localTPF/clientApplication</wsa:Address>
</wsa:From>
<wsa:Action>http://example.com/addressing</wsa:Action>
```

- *Outbound responses*

The WS-Addressing handler will add wsa-addressing elements to the outbound provider XML Encryption responses. The elements are added after the WS-Security Header element.

Open the sample encrypted outbound SOAP response file called SecureCalculatorConsumerResponse.xml and locate the "Address" element:

```
<wsa:From xmlns:wsa="http://www.w3.org/2005/08/addressing">
   <wsa:Address>9.57.13.48</wsa:Address>
</wsa:From>
<wsa:Action>http://example.com/addressing</wsa:Action>
```

- *Changing the wsa-addressing*

1. Open the user exit segment cwscue.c.
2. For outbound consumer requests, edit the code to add your own addressing information to be sent in the request.
3. For outbound provider responses, this sample message handler code uses the host IP address for the "From" "Address". This can be changed to whatever the user desires.

4.3 Key mapping:

The WS-Security SOAP message handler uses keys in the z/TPF secure key keystore to provide support for the encryption and decryption of SOAP messages. Key aliases, which are the key names that appear in actual SOAP message flows to and from z/TPF, are resolved into z/TPF key names based on the client identity (4.1) of the system that is sending a SOAP request.

To get key names needed for decryption of inbound requests and encryption of outbound responses, a user exit `tpf_soapMapKey(client_id, key_alias)` in cso9.c is called with the client system identifier and a key alias name as parameters.

- *Inbound requests:*

For decrypting inbound SOAP requests, the key alias name is retrieved from the <KeyName> element in the request message. (SecureCalculatorConsumerRequest.xml)

Locate the "KeyName" element in the request file that was opened in Step #1:

```
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
   <ds:KeyName>LocalKeyAlias</ds:KeyName>
</ds:KeyInfo>
```

- *Outbound responses:*

For encrypting outbound SOAP responses, key aliases are specified in the extension file. Open the sample WS-Security consumer extension file called calculatorConsumer_ext.xml.

Locate the "KeyAlias" element in the extension file.

```
<ztpfwsse:KeyInformation>
    <ztpfwsse:ExistingKey Id="mykey1">
        <ztpfwsse:KeyAlias>>LocalKeyAlias</ztpfwsse:KeyAlias>
    </ztpfwsse:ExistingKey>
</ztpfwsse:KeyInformation>
```

- *Map the key alias to real key name:*

1. Generate the keys that the consumer and provider Web services will use, import the keys to the z/TPF secure key keystore and activate them. Since this sample application is designed to run with both the consumer and provider on the same z/TPF processors, the keys can be generated using the `ZKEYS` command (Section 7.0). If the consumer and provider were on separate z/TPF processors, the user would need to have a method to import the keys from the other processors.

2. Open the sample user exit cso9.c and see that it contains a table with a client identifier, key alias name, and the 'real' key name (also called the z/TPF secure key keystore name).

   For example, the sample code maps the client identifier of `http://www.ibm.com/localTPF/clientApplication` and the key alias name of `LocalKeyAlias` to the keystore name of `LOCALKEY`. (LOCALKEY must exist in the z/TPF secure key keystore)

3. If you wish to make changes to how the key alias is mapped to the real key name, edit the sample user exit code (cso9.c) to either change the current table to include your specific client identifiers and key alias names or to return the keystore name with a different method.

4. Go to the IBM TPF Product Information Center for more information on z/TPF Security.

4.4 Extension file

The extension file names for the WS-Security SOAP message handler are identified in the Web service deployment descriptor. The values in the SOAP message handler extension files are unique to your organization.

Sample extension file explanations for this WS-Security application:

1. Open the sample Consumer extension file for /CalculatorService (calculatorConsumer_ext.xml)
   - Key aliases are assumed to match actual key names in the z/TPF secure key keystore.
   - Locate the "OutFlow" element. This is used for non-fault responses.

     If the operation that was specified in the inbound SOAP request is `add`, then the "Field" elements under the "EncryptionOutFlow" will be used for encryption.

     In this case, the "/Envelope/Body/add/value" element is listed. It has an encryption type of "Content" which means we will encrypt just the content of the element "value" but not the element tag name "<value>" from its start to end tags.

     The keyId of "mykey1" maps to the "ExistingKey" element (4.3).

```
<ztpfwsse:OutFlow>
    <ztpfwsse:Operation name="add">
        <ztpfwsse:EncryptionOutFlow>
            <ztpfwsse:EncryptFields>
```

```
                <ztpfwsse:Field type="Content" keyId="mykey1">
                     /Envelope/Body/add/value
                </ztpfwsse:Field>
            </ztpfwsse:EncryptFields>
        </ztpfwsse:EncryptionOutFlow>
    </ztpfwsse:Operation>
</ztpfwsse:OutFlow>
```

2. Open the sample Provider extension file for /CalculatorService (calculatorProvider_ext.xml)
   - Key aliases are assumed to match actual key names in the z/TPF secure key keystore.
   - Locate the "OutFlow" element. This is used for non-fault responses.

     If the operation that was specified in the inbound SOAP request is `add`, then the "Field" elements under the "EncryptionOutFlow" will be used for encryption.

     In this case, the "/Envelope/Body/addResponse" element is listed. It has an encryption type of "Element" which means we will encrypt the content of the element "addResponse" AND the element tag name "<addResponse>" from its start to end tags. The identity of the element itself is hidden.

     The keyId of "mykey1" maps to the "ExistingKey" element (4.3).

```
<ztpfwsse:OutFlow>
    <ztpfwsse:Operation name="add">
        <ztpfwsse:EncryptionOutFlow>
            <ztpfwsse:EncryptFields>
                <ztpfwsse:Field type="Element" keyId="mykey1">
                    /Envelope/Body/addResponse
                </ztpfwsse:Field>
            <</ztpfwsse:EncryptFields>
        </ztpfwsse:EncryptionOutFlow>
    </ztpfwsse:Operation>
</ztpfwsse:OutFlow>
```

   - Locate the "Faultflow" element. This is used for fault responses.

     Notice that no Operation is listed. An "Operation" element without a name attribute like `add` serves as a `catch-all` category for all operations not specified in the extension file.

     In this case, the "/Envelope/Body/Fault/faulstring" element is listed. It has an encryption type of "Content" which means we will encrypt the content of the element "faultstring" but not the element tag name "<faultstring>" from its start to end tags.

     The keyId of "mykey1" maps to the "ExistingKey" element (4.3).

```
<ztpfwsse:FaultFlow>
    <ztpfwsse:Operation>
        <ztpfwsse:EncryptionOutFlow>
            <ztpfwsse:EncryptFields>
                <ztpfwsse:Field type="Content" keyId="mykey1">
                    /Envelope/Body/Fault/faultstring
                </ztpfwsse:Field>
            </ztpfwsse:EncryptFields>
        </ztpfwsse:EncryptionOutFlow>
    </ztpfwsse:Operation>
</ztpfwsse:FaultFlow>
```

5.0 COMPILING, LINKING, AND LOADING

1. **cd ~/your_workdir**

2. Create a `maketpf` configuration file named `maketpf.cfg`.
   - Ensure that the first assignment of `TPF_ROOT` in `maketpf.cfg` is the absolute path to your "*`~/your_workdir`*" directory.
   - Ensure that the first assignment of `APPL_ROOT` in `maketpf.cfg` is the absolute path to your "*`~/your_workdir`*" directory.
   - Update other fields (`TPF_BSS_NAME, TPF_SS_NAME, USER_VERSION_CODE`) if necessary.

3. Compile and link the WS-Security user exit sample programs and message handler program.

   maketpf cso9.mak -f
   maketpf cwsu.mak -f
   maketpf csok.mak -f

4. Compile and link the SOAP provider sample programs.

   maketpf csob.mak -f

5. Use the standard load procedure to transfer and load the SOAP sample programs (CSO9, CWSU, CSOK, CSOB) to your test system.

6.0 DEPLOYING

To deploy the sample Web services, making them accessible to the z/TPF SOAP handler, you will need to FTP the consumer and provider Web service deployment descriptors and SOAP message handler descriptors to your z/TPF system and use the ZWSAT DEPLOY command. You will also need to define a mechanism for activating the sample SOAP consumer application (QWC2) which was part of the download in Step 2.3.

The sample is designed to run with both the SOAP consumer and SOAP provider on the same z/TPF processor. For example, the transport definitions in the consumer Web service deployment descriptor make use of the well-known loopback IP address (127.0.0.1) or rely on queue definitions that are local to the current z/TPF processor.

If you wish to run the sample with the SOAP consumer and SOAP provider on separate z/TPF processors, update the consumer Web service deployment descriptor accordingly before continuing with these deployment steps.

1. FTP the Web service deployment descriptors and SOAP message handler descriptors to the `/etc/tpf-ws/` directory on your z/TPF system using binary mode:
   - Change to the "*`~/your_workdir`*" directory:
     cd ~/your_workdir
   - FTP by using the following command:
     ftp *`your.zTPF.system`*
   - Sign in using your user name and password.
   - Set the mode to binary by entering the following command:
     binary
   - Send the file to your z/TPF system by using the following command:

     send SecureCalculatorService_consumer.xml /etc/tpf-ws/CalculatorService_consumer.xml

     send SecureCalculatorService.xml /etc/tpf-ws/CalculatorService.xml

     send WS-Security.xml /etc/tpf-ws/WS-Security.xml

     send WS-Addressing.xml /etc/tpf-ws/WS-Addressing.xml

2. FTP the Web service provider and consumer WS-Security extension files to the `/etc/tpf-ws/ext/WS-Security` directory on your z/TPF system using binary mode:
   - Send the file to your z/TPF system by using the following command:

     send calculatorConsumer_ext.xml /etc/tpf-ws/ext/WS-Security/calculatorConsumer_ext.xml

     send calculatorProvider_ext /etc/tpf-ws/ext/WS-Security/calculatorProvider_ext.xml

   - Exit FTP by entering the following command:
     bye

3. On your z/TPF system, deploy the WS-Security SOAP Message Handlers and consumer and provider Web services by entering the following commands:

   ZWSAT DEPLOY DD-WS-Security.xml

   ZWSAT DEPLOY DD-WS-Addressing.xml

   ZWSAT DEPLOY DD-SecureCalculatorService_consumer.xml

   ZWSAT DEPLOY DD-SecureCalculatorService.xml

4. On your z/TPF system, verify the deployment of the descriptors and display the extension file data:

   ZWSAT DISPLAY DD-SecureCalculatorService_consumer.xml

   ZWSAT DISPLAY DD-SecureCalculatorService_consumer.xml ext-all

   ZWSAT DISPLAY DD-SecureCalculatorService.xml

   ZWSAT DISPLAY DD-SecureCalculatorService.xml ext-all

5. Ensure that the communications binding(s) that you intend to use are running.

7.0 GENERATING KEYS

1. Generate and activate the key in the z/TPF secure key keystore using the `ZKEYS` command. Go to the IBM TPF Product Information Center for more details on ZKEYS.

   ZKEYS GENERATE ENC-LOCALKEY DEC-LOCALKEY CIPHER-AES128CBC NEW

   ZFILE mkdir /your_backup_dir

   ZKEYS BACKUP PATH-/your_backup_dir/your_backup_filename PASSWORD-your_backup_password

   ZKEYS ACTIVATE ENC-LOCALKEY DEC-LOCALKEY

8.0 RUNNING

The WS-Security SOAP consumer sample application demonstrates the use of Web Services Security (XML Encryption) on z/TPF.

The application invokes the `/CalculatorService` sample Web service on z/TPF.
The `/CalculatorService` exposes two operations: `add` and `sub`. You can specify which operation to invoke, along with the two values that are to be added or subtracted.

The expected input format for the sample SOAP consumer application is as follows:

ZCALC *operation* VAL1-*value1* VAL2-*value2* [Transport-*transport*] [Mep-*mep*] [TImeout-*to*]

For example:

ZCALC add VAL1-6 VAL2-77 TRANSPORT-HTTP MEP-SYNC

The response should look like this:

```
CALC0100I 16.52.22 RESPONSE RECEIVED FOR /CalculatorService

  Synchronous call to add operation was successful
  HTTP transport was used
  Result of 100 plus 400 is 500

END OF DISPLAY
```

The fact that encryption was used is transparent to the application, so the results of running the command will not indicate that encryption was used in any way.

9.0 NOTICES

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service. IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department 31BA/Building 008
Mail Drop P369
2455 South Road
Poughkeepsie, NY 12601-5400
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee. Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

8.1 Trademarks

IBM is a trademark of International Business Machines Corporation in the United States, other countries, or both.

Microsoft is a registered trademark of Microsoft Corporation in the United States, other countries, or both.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.